修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工　学研究科 情報・ネットワーク工学 専攻 博士前期課程		
氏　　名	中山大輔	学籍番号	1631107
論 文 題 目	拡張 affine 演算を用いた精度保証計算ライブラリ		

要　旨

本論文では、精度保証付き数値計算における affine 演算を拡張した新しい算法の提案・実装を行い、その算法が有用な場面について論じる。

精度保証付き数値計算で用いられる区間演算の算法やライブラリは、一般に区間幅が狭いことを前提に設計されており、入力の区間幅が大きいと意味のある計算結果を返さない場合がある。そのため、入力の区間幅がある程度の大きさを持つ場合は何らかの工夫をする必要があり、簡単なものでは入力の区間を分割するということが考えられる。なお、多倍長演算を用いることも考えられるが、これは丸め誤差を軽減させるためのものなので、「入力の区間幅が大きい場合でも意味のある出力を得られるようにする」という目的には適していない。

近年、高次元の問題への精度保証付き数値計算の応用が行われているが、入力区間の分割は入力変数の個数に関して指数関数的な計算時間の増加をもたらすので、高次元の問題ではできる限り入力区間の分割を行わずに計算ができる必要がある。区間の間の相関を考慮することで計算時の区間拡大を抑える演算で affine 演算というものがあるが、これは非線形な演算を一次式で近似するため、割り算などの演算に対して精度が出にくい場合がある。そこで、affine 演算を拡張し、非線形な演算を二次式で近似することを考える。

affine 演算を拡張したものを「拡張 affine 演算」と呼ぶことにし、拡張 affine 演算における四則演算を実装したライブラリを作成した。数値実験により、出力の精度が一定以下になることが求められていて入力を分割しなければならないような状況では、affine 演算よりも拡張 affine 演算の方が速度の面で優れている場合があり、特に入力が多変数となると、拡張 affine 形式の方が数百倍程度早く計算できる場合があることがわかった。

平成 29 年度　修士論文

拡張 affine 演算を用いた精度保証計算ライブラリ

学籍番号 1631107
中山大輔
情報・ネットワーク工学専攻
主任指導教員:山本野人教授
指導教員:山﨑匡准教授

目次

1 はじめに

精度保証付き数値計算で使われる区間演算の算法やライブラリは、一般に区間幅が小さいことを前提に設計されており、入力の区間幅が大きいと意味のある結果を返さない場合がある。そのため、区間の情報を多倍長浮動小数点数で保持することで丸め誤差の影響を軽減させたり、区間幅が大きい場合は区間を分割するということが行われる。

近年は高次元の問題への精度保証付き数値計算の適用が行われようとしているが、区間の分割は入力の数に関して指数関数的な計算量の増加を引き起こすので、高次元の問題では、ある程度の幅を持つ区間に対してもできる限り区間の分割を行わず計算を行う必要がある。多倍長浮動小数点数ではこのような目的には対応できないので、ある程度の大きさを持つ区間に対しても精度を保ったまま計算できるようにするには算法の工夫が必要となる。区間の間の相関を考慮することである程度の大きさを持つ区間に対しても精度を保って計算できるような算法で affine 演算というものがあるが、これは非線形な演算を一次式で近似するため、割り算などの演算で精度が出にくい場合がある。そこで本論文では、affine 演算を拡張して、非線形な演算に対しても精度が出やすい算法と、それを実装したライブラリを開発することを目的とする。

2 精度保証付き数値計算の技法

2.1 区間演算

区間演算について、[1] に基づいて説明する。実数の区間 $[a,b]$ とは次のような $\mathbb{R}$ の閉集合である。

$$[a,b] = \{\, x \mid a \le x \le b \,\}.$$

区間に対する演算は、区間に含まれる任意の要素に対する演算結果をすべて包含するような最小の有界な閉区間として定義される。二つの区間 $X = [a,b]$, $Y = [c,d]$ に対する四則演算は以下のようになる。

$$\begin{aligned}
X + Y &= [a + c, b + d], \\
X - Y &= [a - d, b - c], \\
X \cdot Y &= [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}], \\
X/Y &= [a,b] \cdot [1/d, 1/c],\ 0 \notin Y
\end{aligned}$$

計算機上で実装する際には、浮動小数点数演算の丸め方向制御を用いて上端を上向き、下端を下向きに丸めるなどして、真の結果を包含するような区間が得られるようにする。

区間演算では分配法則が成り立たず、半分配則 (劣分配則) しか成り立たないことに注意が必要である。すなわち、A, B, C を区間とすると、

$$A(B + C) \subset AB + AC$$

である。また、区間演算では一般に

$$\begin{aligned}
X - X &\neq [0,0], \\
X/X &\neq [1,1]
\end{aligned}$$

であることにも注意が必要である。

区間 $X = [a, b]$ の半径を $\mathrm{rad}(X) \equiv (b-a)/2$、中心を $\mathrm{mid}(X) \equiv (a+b)/2$ と定義する。また、$\mathrm{mag}(X) \equiv \max\{|a|, |b|\}$ とする。

区間を並べた行列やベクトルを区間行列や区間ベクトルと呼ぶ。また、区間行列や区間ベクトルに対する mid, rad, mag を、各要素に mid, rad, mag を適用したものとして定義する。

区間演算をそのまま適用したのでは区間幅が大きく拡大する場合があるが、その原因は大きく分けて二つある。一つは dependency problem である。これは分配律が成り立たないことに起因する区間拡大で、非線形関数の包含でよく生じる。これを軽減する方法は、平均値形式と呼ばれる方法やそのバリエーションがある。もう一つの原因は wrapping effect で、これは行列と区間ベクトルの積によって区間ベクトルが回転作用と歪み作用を受け、その結果を区間で包含する際に発生する区間拡大である。これを軽減する方法は計算順序の変更や座標変換などがある。以下で述べる affine 演算は、dependency problem および wrapping effect に対して、ともに軽減効果がある。

2.2 affine 演算

affine 演算について、[3] に基づいて説明する。これは変数間の相関を考慮することで区間演算の過大評価を防ぐ方法の一つで、変動範囲が $[-1, 1]$ であるようなダミー変数 ε_i の線形結合

$$x_0 + x_1\varepsilon_1 + \cdots + x_k\varepsilon_k$$

の形 (affine 形式) によって区間を表現する。ダミー変数の数 k は計算の途中で変化する。

2 つの affine 形式 x, y に相関がない場合は、例えば

$$\begin{aligned} x &= 1 + 0.5\varepsilon_1, \\ y &= 1 + 0.5\varepsilon_2 \end{aligned}$$

のようになり、相関がある場合は

$$\begin{aligned} x &= 1 + 0.5\varepsilon_1, \\ y &= 1 + 0.4\varepsilon_1 + 0.1\varepsilon_2 \end{aligned}$$

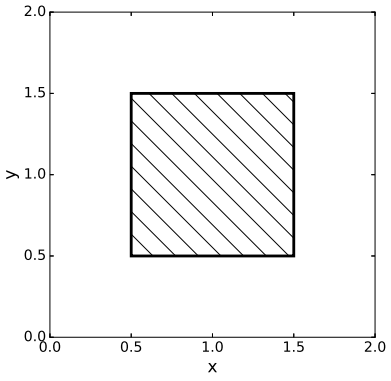
のようになる。x, y それぞれが取り得る範囲はいずれも $[0.5, 1.5]$ だが、ダミー変数をそれぞれ -1 から 1 まで動かして得られる領域は異なる (図 1)。

n 変数関数 $f(x_1, \ldots, x_n)$ を affine 演算で評価する場合、各入力変数の変域を $[\underline{x}_i, \overline{x}_i]$ $(i = 1, \ldots, n)$ とすると、n 個のダミー変数を用いて

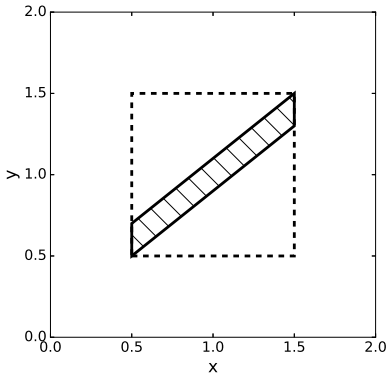
$$\begin{aligned} x_1 &= \frac{\overline{x}_1 + \underline{x}_1}{2} + \frac{\overline{x}_1 - \underline{x}_1}{2}\varepsilon_1, \\ x_2 &= \frac{\overline{x}_2 + \underline{x}_2}{2} + \frac{\overline{x}_2 - \underline{x}_2}{2}\varepsilon_2, \\ &\vdots \\ x_n &= \frac{\overline{x}_n + \underline{x}_n}{2} + \frac{\overline{x}_n - \underline{x}_n}{2}\varepsilon_n \end{aligned}$$

のような affine 形式で初期化してから評価する。

与えられた affine 形式 $y_0 + y_1\varepsilon_1 + \cdots + y_k\varepsilon_k$ から区間への変換をするには、中心を y_0、半径を $\sum_{i=1}^{k} \left|y_i\right|$ とする区間を作れば良い。



(a) x, y に相関がない場合



(b) x, y に相関がある場合

図 1: x, y の相関の有無と領域の関係

2.2.1 四則演算

affine 形式に対する定数倍や、affine 形式同士の加減算は、単に各係数を定数倍したり、係数ごとに加減算を行えば良い。乗算は、$x = x_0 + x_1\varepsilon_1 + \cdots + x_k\varepsilon_k$, $y = y_0 + y_1\varepsilon_1 + \cdots + y_k\varepsilon_k$ とおくと、

$$xy \in x_0y_0 + \sum_{i=1}^{k}(y_0x_i + x_0y_i)\varepsilon_i + \left(\sum_{i=1}^{k}|x_i|\right)\left(\sum_{i=1}^{k}|y_i|\right)\varepsilon_{k+1} \tag{1}$$

とする場合が多い。これ以外の方法では、例えば [4] で誤差項を最小にする「最良乗算」が提案されている。除算は、$x \times (1/y)$ と逆数関数と乗算に分解することが多い。

2.2.2 非線形単項演算

非線形な単項演算は、演算 $f(x)$ を線形な演算で近似し、その誤差を新しいダミー変数 ε_{k+1} で表すようにする。具体的には、$x = x_0 + x_1\varepsilon_1 + \cdots + x_k\varepsilon_k$ を区間に変換したものを $I = [\underline{x}, \overline{x}]$ とおくと、

$$\delta = \max_{x \in I}\left|f(x) - (px + q)\right|$$

を満たす p, q, δ を δ ができるだけ小さくなるように求め、

$$p(x_0 + x_1\varepsilon_1 + \cdots + x_k\varepsilon_k) + q + \delta\varepsilon_{k+1}$$

を結果とする。f が I 内で凸の場合は、

$$p = \frac{f(\overline{x}) - f(\underline{x})}{\overline{x} - \underline{x}}$$

として、$\xi \in I$ が $f'(\xi) = p$ を満たすとすると、

$$q = \frac{f(\underline{x}) + f(\xi) - p(\underline{x} + \xi)}{2},$$
$$\delta = \left|\frac{f(\xi) - f(\underline{x}) - p(\xi - \underline{x})}{2}\right|$$

とすれば、δ が最小となる p, q, δ を求めることができる [5]。

2.2.3 丸め誤差の処理

計算機上で実装する際には、係数の計算などで丸め誤差が発生する。対処法の一つとして丸め誤差を格納する専用のダミー変数を用意するものがある。これは、次のように各 affine 形式にダミー変数 ε_r を加える方法である。

$$x_0 + x_1\varepsilon_1 + \cdots + x_k\varepsilon_k + \delta_r\varepsilon_r,\ \delta_r \geq 0.$$

丸め誤差が発生しないうちは $\delta_r = 0$ としておく。例えば、2 つの affine 形式 $x = x_0 + x_1\varepsilon_1 + \cdots + x_k\varepsilon_k + \delta_x\varepsilon_{rx}$, $y = y_0 + y_1\varepsilon_1 + \cdots + y_k\varepsilon_k + \delta_y\varepsilon_{ry}$ の加算の場合は、

$$\begin{aligned} x + y \in & \underbrace{(x_0 + y_0)}_{=:z_0} + \underbrace{(x_1 + y_1)}_{=:z_1}\varepsilon_1 + \cdots + \underbrace{(x_k + y_k)}_{=:z_k}\varepsilon_k + \underbrace{(\delta_x + \delta_y)}_{=:\delta_z}[-1, 1] \\ & =: z \end{aligned}$$

とおくと、$z_0, \ldots, z_k, \delta_z$ の計算で丸め誤差が発生することになる。$z_0, \ldots, z_k, \delta_z$ の計算値を $z_0', \ldots, z_k', \delta_z'$ とおき、丸め誤差を $\Delta z_0, \ldots, \Delta z_k, \Delta\delta_z$ とおくと、

$$\begin{aligned} z &= (z_0' + \Delta z_0) + (z_1' + \Delta z_1)\varepsilon_1 + \cdots + (z_k' + \Delta z_k)\varepsilon_k + (\delta_z + \Delta\delta_z)[-1, 1] \\ &\subset z_0' + z_1'\varepsilon_1 + \cdots + z_k'\varepsilon_k + \underbrace{\left(\delta_z + |\Delta\delta_z| + \sum_{i=0}^{k} |\Delta z_i|\right)}_{=:\delta_z''}[-1, 1] \end{aligned}$$

と計算できる。最後の項をダミー変数 ε_{rz} で表すと、

$$z_0' + z_1'\varepsilon_1 + \cdots + z_k'\varepsilon_k + \delta_z''\varepsilon_{rz}$$

が結果となる。非線形な演算の場合は、見積もった丸め誤差を、打切り誤差のために新しく追加するダミー変数 ε_{k+1} の係数に加えて、ε_r の係数は 0 にする。丸め誤差の推定には、係数を区間演算を用いて計算し、中心を計算値、半径を丸め誤差とする方法などがある。

3 多項式近似

3.1 交代定理 [6]

区間 $[a, b]$ 上で定義された連続関数 $f(x)$ を、多項式 $P(x) = c_n x^n + c_{n-1}x^{n-1} + \cdots + c_0$ で近似する問題を考える。

$$\max_{a \leq x \leq b} \left| f(x) - P(x) \right|$$

を最小にする多項式を最良近似という。また、$\sum_{i=0}^{n} c_i g_i(x)$ の形をした関数を意味する一般化された多項式についても、同様な近似を考えることができる。交代定理は、$P(x)$ が最良近似であるための必要十分条件を与える。

定義 1 (Haar の条件) 関数系 $\{g_1, \ldots, g_n\}$ は、もし各 g_i が連続で、

$$\hat{x} = [g_1(x), \ldots, g_n(x)]$$

の形をした n 個のいかなるベクトルも一次独立ならば、Haar の条件を満たすという。言い換えると、n 個の異なる点 $x_1, x_2, \ldots, x_n$ からつくられた行列式

$$\begin{vmatrix} g_1(x_1) & \cdots & g_n(x_1) \\ \vdots & \ddots & \vdots \\ g_1(x_n) & \cdots & g_n(x_n) \end{vmatrix}$$

が 0 でないという条件である。

異なる点に対する Vandermonde の行列式

$$\begin{vmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 & \cdots & x_{n+1}^n \end{vmatrix} = \prod_{1 \le i < j \le n+1} (x_i - x_j)$$

は 0 でないので、$\{1, x, x^2, \ldots, x^n\}$ はいかなる区間に対しても Haar の条件を満たす。Haar の条件を満たす関数系は、しばしば Chebyshev 系と呼ばれる。

定理 2 (交代定理) $\{g_1, \ldots, g_n\}$ を Haar の条件を満たす $C[a,b]$ 上の要素の系とし、$X \subset [a,b]$ を任意の閉集合とする。このとき、一般化された多項式 $p(x) = \sum_{i=i}^{n} c_i g_i(x)$ が X 上で与えられた $f \in C[X]$ の最良近似であるためには、誤差関数 $e(x) = f(x) - p(x)$ が $m+1 > n$ 個の点 $x_0 < \cdots < x_m$ で

$$\left|e(x_i)\right| = \|e\| \quad (i = 0, 1, \ldots, m)$$

となって、かつ

$$e(x_i) = -e(x_{i-1}) \quad (i = 1, 2, \ldots, m)$$

のように $e(x_i)$ の符号が交替することが必要かつ十分である。ここで、$\|e\| = \max_{x \in X} \left|e(x)\right|$ である。

3.2 Remes の第二アルゴリズム [6]

Remes の第二アルゴリズムは、関数 $e(x) = f(x) - \sum_{j=1}^{n} c_j g_j(x)$ の一様ノルムを $[a,b]$ 上で最小にする係数 $c_1, \ldots, c_n$ を求めるアルゴリズムである。ここでは、$\{g_1, \ldots, g_n\}$ が Haar の条件を満たすとする。

アルゴリズムのそれぞれのサイクルで、前のサイクルの順序付きの $n+1$ 個の点の集合 $a \le x_0 < \cdots < x_n \le b$ が与えられている (最初のステップでは、この点は任意に選べば良い)。まず、$\left|e(x_i)\right|$ が等しく、$e(x_i) = -e(x_{i-1})$ $(i = 1, 2, \ldots, n)$ となるように係数 $c_1, c_2, \ldots, c_n$ を求める。中間値の定理から $e(x)$ はそれぞれの区間 (x_{i-1}, x_i) に根 z_i を持つ。さらに、$z_0 \coloneqq a$, $z_{n+1} \coloneqq b$ とおく。また、$\sigma_i \coloneqq \operatorname{sgn} e(x_i)$ とする。それぞれの $i = 0, \ldots, n$ に対し、$\sigma_i e(y)$ が最大となる点 y_i を $[z_i, z_{i+1}]$ の中に選び、試みの集合 $\{y_0, \ldots, y_n\}$ を決める。何らかの方法で $\|e\|$ を評価し、もし $\|e\| > \max_i \left|e(y_i)\right|$ ならば、$\{y_0, \ldots, y_n\}$ の定義を次のように変更する。

- y を $\left|e(y)\right|$ が最大となる点とする。
- y を $\{y_0, \ldots, y_n\}$ の正しい位置に入れ、得られた順序集合上で、e がなおも符号をかえるように一つの y_i を取り除く。

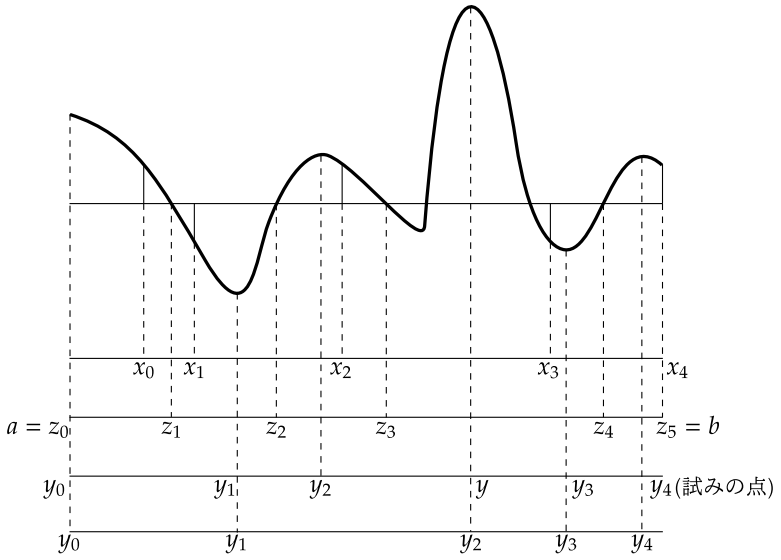


図 2: $n = 3$ のときの例

次のサイクルでは、$\{x_0, \ldots, x_n\}$ の代わりに $\{y_0, \ldots, y_n\}$ から始める。

$n = 4$ のときの、誤差関数のグラフと点の取り方の例を図 2 に示す。

[7] では、f, g_1, $\ldots$, g_n が区間 $[a, b]$ 上で連続かつ、これらの導関数が区間 (a, b) 上で連続で、最良近似の誤差関数の極値が $[a, b]$ の中にちょうど $n + 1$ 個あるとき、このアルゴリズムが 2 次的に収束することが示されている。

3.3 Chebyshev 補間

ある関数 $f(x)$ に対し、相異なる点 $x_0, x_1, \ldots, x_k$ と関数値 $f(x_0), f(x_1), \ldots, f(x_k)$ が与えられたとき、それ以外の点 x (普通は $x_0 < x < x_k$) での関数値を推測することを補間という [8]。また、$(x_i, f(x_i))$ $(i = 0, 1, \ldots, k)$ を補間点という。

次の T_k を、k 次の第一種 Chebyshev 多項式という。

$$T_k(\cos\theta) = \cos k\theta. \tag{2}$$

Chebyshev 多項式は $-1 \le x \le 1$ ならば $-1 \le T_k(x) \le 1$ で、$[-1, 1]$ 内に k 個のゼロ点を持つ。ゼロ点は次の式で与えられる。

$$x = \cos\left(\frac{\pi\left(i - \frac{1}{2}\right)}{k}\right) \quad (i = 1, 2, \ldots, k).$$

補間点として Chebyshev 多項式のゼロ点を使うとき、$f(x)$ が区間 $[-1, 1]$ で正則な解析関数ならば、次数を上げると区間全体で良い近似を得ることができる [8]。

近似の範囲を $[-1, 1]$ から別の区間 $[a, b]$ に変更する場合は、まず

$$x = \frac{x' - \frac{1}{2}(a + b)}{\frac{1}{2}(b - a)}$$

と変数変換する。すると、

$$T_k(x) = T_k\left(\frac{x' - \frac{1}{2}(a+b)}{\frac{1}{2}(b-a)}\right),\ x' \in [a,b]$$

となる。よって、

$$x_i = \cos\left(\frac{\pi\left(i-\frac{1}{2}\right)}{k}\right) \quad (i = 1,\ 2,\ \ldots,\ k),$$

$$x'_i = \cos\left(\frac{\pi\left(i-\frac{1}{2}\right)}{k}\right)\frac{b-a}{2} + \frac{a+b}{2} \quad (i = 1,\ 2,\ \ldots,\ k) \tag{3}$$

となるので、x'_i を補間点として用いれば良い。

$f(x)$ を $[-1,1]$ の任意の関数として、x_i $(i = 1,\ 2,\ \ldots,\ k)$ を $T_k(x)$ のゼロ点とする。c_i $(i = 1,\ 2,\ \ldots,\ k-1)$ を次のように定める。

$$\begin{aligned} c_i &= \frac{2}{k}\sum_{j=1}^{k} f(x_j)T_i(x_j) \\ &= \frac{2}{k}\sum_{j=1}^{k} f\left(\cos\left(\frac{\pi\left(j-\frac{1}{2}\right)}{k}\right)\right)\cos\left(\frac{\pi i\left(j-\frac{1}{2}\right)}{k}\right) \end{aligned}$$

このとき、$f(x)$ の近似多項式を

$$f(x) \approx \left(\sum_{j=0}^{k} c_j T_j(x)\right) - \frac{1}{2}c_0$$

と定める。これは $T_k(x)$ の k 個のゼロ点で $f(x)$ と一致する [9]。区間を $[-1,1]$ から別の区間 $[a,b]$ に変更する場合は、c_i を

$$c_i = \frac{2}{k}\sum_{j=1}^{k} f\left(\frac{b-a}{2}x_j + \frac{a+b}{2}\right)T_i(x_j) \tag{4}$$

と定め、近似多項式を

$$f(x) \approx \left(\sum_{j=0}^{k} c_j T_j\left(\frac{x - \frac{1}{2}(a+b)}{\frac{1}{2}(b-a)}\right)\right) - \frac{1}{2}c_0 \tag{5}$$

とする [9]。

4 affine 演算の拡張

ここでは、ダミー変数について二次の項を加えることで affine 演算を拡張することを考える。まず、affine 形式を次のようにベクトルの内積を使って表現する。

$$x_0 + x_1\varepsilon_1 + \cdots + x_k\varepsilon_k + \delta_x\varepsilon_{rx} = x_0 + \boldsymbol{x}^T\boldsymbol{\varepsilon}_k + \delta_x\varepsilon_{rx},$$

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix},\ \boldsymbol{\varepsilon}_k = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_k \end{pmatrix},\ \varepsilon_i \in [-1,1] \quad (i = 1,\ \ldots,\ k).$$

そして、次のように二次形式を付け加えることで二次の項を追加する。

$$x_0 + \boldsymbol{x}^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k + \delta_x \varepsilon_{rx}. \tag{6}$$

ただし、X は $k \times k$ 実対称行列とする。ここでは、この式を用いた演算を「拡張 affine 演算」と呼ぶことにする。この章では、この式を区間に変換するために必要な計算と、拡張 affine 演算における四則演算について述べる。

4.1 一次項の評価

(6) の値域を計算するときなどに一次項の評価が必要になる。$\boldsymbol{x}$ の要素を x_i, $i = 1, \ldots, k$ とおくと、

$$\boldsymbol{x}^T\boldsymbol{\varepsilon}_k = \sum_{i=1}^{k} x_i \varepsilon_i \in \left(\sum_{i=1}^{k} |x_i| \right) [-1, 1] \tag{7}$$

となる。これは affine 形式を区間に変換するときに使う式と同じである。右辺を評価して得られる区間を $[\boldsymbol{x}^T\boldsymbol{\varepsilon}_k]$ と書く。

4.2 二次項の評価

(6) を区間に変換するときなどには、二次項の評価も必要になる。ここでは、X が実対称であることと、ダミー変数の積が

$$\varepsilon_i \varepsilon_j \in \begin{cases} [0, 1] & \text{if } i = j \\ [-1, 1] & \text{if } i \neq j \end{cases}$$

と区間で包含できることを利用し、

$$\boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k = \sum_{i=1}^{k} x_{ii} \varepsilon_i^2 + \sum_{i=1}^{k} \sum_{j=i+1}^{k} 2 x_{ij} \varepsilon_i \varepsilon_j \in \sum_{i=1}^{k} x_{ii} [0, 1] + \sum_{i=1}^{k} \sum_{j=i+1}^{k} 2 x_{ij} [-1, 1] \tag{8}$$

と評価する。ただし、x_{ij} は X の (i, j) 成分である。

$\boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k$ を (8) の右辺で評価して得られる区間を $[\boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k]$ と書く。

4.3 四則演算

4.3.1 加算

$x \coloneqq x_0 + \boldsymbol{x}^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k + \delta_x \varepsilon_{rx}$, $y \coloneqq y_0 + \boldsymbol{y}^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Y \boldsymbol{\varepsilon}_k + \delta_y \varepsilon_{ry}$ とおくと、拡張 affine 演算における加算は次のようになる。

$$\begin{aligned} x + y &\in \underbrace{(x_0 + y_0)}_{=: z_0} + \underbrace{(\boldsymbol{x} + \boldsymbol{y})}_{=: \boldsymbol{z}}{}^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T \underbrace{(X + Y)}_{=: Z} \boldsymbol{\varepsilon}_k + \underbrace{(\delta_x + \delta_y)}_{=: \delta_z} [-1, 1] \\ &=: z. \end{aligned}$$

計算機上で実装する際には、$x_0 + y_0$ や $\boldsymbol{x} + \boldsymbol{y}$ などの各演算で丸め誤差が発生する。z の計算値を $z_0' + \boldsymbol{z}'^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k + \delta_z' [-1, 1]$ とおき、丸め誤差による真値とのずれを

$$z = (z_0' + \Delta z_0) + (\boldsymbol{z}' + \Delta \boldsymbol{z})^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T (Z' + \Delta Z) \boldsymbol{\varepsilon}_k + (\delta_z' + \Delta \delta_z)[-1, 1]$$

と表現する。Δz_0, $\Delta \boldsymbol{z}$, ΔZ, $\Delta \delta_z$ が得られたとすると、z は

$$\begin{aligned} z &= (z_0' + \Delta z_0) + (\boldsymbol{z}' + \Delta \boldsymbol{z})^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T (Z' + \Delta Z) \boldsymbol{\varepsilon}_k + (\delta_z' + \Delta \delta_z)[-1, 1] \\ &\subset z_0' + \boldsymbol{z}'^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k + (\delta_z' + \Delta \delta_r + \Delta z_0)[-1, 1] + [\Delta \boldsymbol{z}^T \boldsymbol{\varepsilon}_k] + [\boldsymbol{\varepsilon}_k^T \Delta Z \ \boldsymbol{\varepsilon}_k] \\ &\subset z_0' + \boldsymbol{z}'^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k + \underbrace{(\delta_z' + \Delta \delta_r + \Delta z_0 + \mathrm{mag}([\Delta \boldsymbol{z}^T \boldsymbol{\varepsilon}_k]) + \mathrm{mag}([\boldsymbol{\varepsilon}_k^T \Delta Z \ \boldsymbol{\varepsilon}_k]))}_{=:\delta_z''}[-1, 1] \end{aligned}$$

と評価できる。新しいダミー変数 ε_{rz} で最後の項を表すことにすると、

$$z_0' + \boldsymbol{z}'^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k + \delta_z'' \varepsilon_{rz}$$

で加算を表すことができる。Δz_0, $\Delta \boldsymbol{z}$, ΔZ, $\Delta \delta_z$ を得るには、例えば計算を区間演算で行い、結果の中心を z_0', $\boldsymbol{z}'$, Z', δ_z'、半径を Δz_0, $\Delta \boldsymbol{z}$, ΔZ, $\Delta \delta_z$ とする方法がある。

4.3.2 減算

減算は次のように行う。

$$x - y \in (x_0 - y_0) + (\boldsymbol{x} - \boldsymbol{y})^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T (X - Y) \boldsymbol{\varepsilon}_k + (\delta_x + \delta_y)[-1, 1].$$

計算機で実装する際には、各演算で丸め誤差が発生するので、加算の場合と同様に処理をする。

4.3.3 乗算

乗算は次のように行う。

$$\begin{aligned} xy &= (x_0 + \boldsymbol{x}^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k + \delta_x \varepsilon_{rx})(y_0 + \boldsymbol{y}^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Y \boldsymbol{\varepsilon}_k + \delta_y \varepsilon_{ry}) \\ &\in \underbrace{x_0 y_0}_{=:z_0} + \underbrace{(x_0 \boldsymbol{y} + y_0 \boldsymbol{x})}_{=:\boldsymbol{z}}{}^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T \underbrace{(\boldsymbol{x}\boldsymbol{y}^T + x_0 Y + y_0 X)}_{=:Z} \boldsymbol{\varepsilon}_k \\ &\quad + \boldsymbol{\varepsilon}_k^T \underbrace{(\boldsymbol{x}\boldsymbol{\varepsilon}_k^T Y + \boldsymbol{y}\boldsymbol{\varepsilon}_k^T X + X \boldsymbol{\varepsilon}_k \boldsymbol{\varepsilon}_k^T Y)}_{\in [A]\text{：区間行列}} \boldsymbol{\varepsilon}_k + (\delta_x y + \delta_y (x - \delta_x \varepsilon_{rx}))[-1, 1] \\ &\subset z_0 + \boldsymbol{z}^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T \underbrace{(Z + \mathrm{mid}([A]))}_{=:Z'} \boldsymbol{\varepsilon}_k + [\boldsymbol{\varepsilon}_k^T ([A] - \mathrm{mid}([A])) \boldsymbol{\varepsilon}_k] + \underbrace{\mathrm{mag}(\delta_x y + \delta_y (x - \delta_x \varepsilon_{rx}))}_{=:\delta_z} [-1, 1] \end{aligned}$$

δ_z は、x, y を (7), (8) を用いて評価することで計算する。また、区間行列 $[A]$ は、$\boldsymbol{\varepsilon}_k$, $\boldsymbol{\varepsilon}_k \boldsymbol{\varepsilon}_k^T$ を

$$\begin{gathered} \varepsilon_i \in [-1, 1] \quad (i = 1, \ \ldots, \ k), \\ \varepsilon_i \varepsilon_j \in \begin{cases} [0, 1] & \text{if } i = j \\ [-1, 1] & \text{if } i \neq j \end{cases} \quad (i, j = 1, \ \ldots, \ k) \end{gathered}$$

を利用して区間行列に置き換えることで計算する。すると、乗算は

$$\begin{aligned} xy &\in z_0 + \boldsymbol{z}^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k + [\boldsymbol{\varepsilon}_k^T ([A] - \mathrm{mid}([A])) \boldsymbol{\varepsilon}_k] + \delta_z [-1, 1] \\ &= z_0 + \boldsymbol{z}^T \boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k + (\mathrm{rad}([\boldsymbol{\varepsilon}_k^T ([A] - \mathrm{mid}([A])) \boldsymbol{\varepsilon}_k]) + \delta_z)[-1, 1] \end{aligned}$$

となる。Z' が一般には対称行列にならないので、

$$\begin{aligned} \boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k &= \frac{(\boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k) + (\boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k)^T}{2} \\ &= \boldsymbol{\varepsilon}_k^T \frac{Z' + Z'^T}{2} \boldsymbol{\varepsilon}_k \end{aligned}$$

として対称にし、最後の項を新しいダミー変数 ε_{k+1} と ε_{rz} を用いて表すと、

$$z_0 + \begin{pmatrix} \boldsymbol{z} \\ \mathrm{rad}([\boldsymbol{\varepsilon}_k^T([A] - \mathrm{mid}([A]))\boldsymbol{\varepsilon}_k]) + \delta_z \end{pmatrix}^T \boldsymbol{\varepsilon}_{k+1} + \boldsymbol{\varepsilon}_{k+1}^T \begin{pmatrix} \frac{Z' + Z'^T}{2} & \begin{matrix} 0 \\ \vdots \end{matrix} \\ \begin{matrix} 0 & \cdots \end{matrix} & 0 \end{pmatrix} \boldsymbol{\varepsilon}_{k+1} + 0\varepsilon_{rz}$$

で乗算を表すことができる。

$[\boldsymbol{\varepsilon}_k^T([A] - \mathrm{mid}([A]))\boldsymbol{\varepsilon}_k]$ の計算法を示す。$[A] - \mathrm{mid}([A])$ の各要素がゼロ中心の区間で、ゼロ中心の区間 $[-r, r]$ に対して

$$[-r, r] \cdot [0, 1] = [-r, r],$$
$$[-r, r] \cdot [-1, 1] = [-r, r]$$

で、ダミー変数に対して

$$\varepsilon_i \varepsilon_j \in \begin{cases} [0, 1] & \text{if } i = j \\ [-1, 1] & \text{if } i \neq j \end{cases}$$

となるので、$[A] - \mathrm{mid}([A])$ の (i, j) 成分を $[a_{ij}]$ とすると、

$$[\boldsymbol{\varepsilon}_k^T([A] - \mathrm{mid}([A]))\boldsymbol{\varepsilon}_k] \subset \sum_{i=1}^{k} \sum_{j=1}^{k} [a_{ij}]$$

とできる。

各演算での丸め誤差を評価する必要があるのは加算や減算と同様だが、求めた誤差は ε_{k+1} の係数に加える。

4.3.4 除算

除算は $x/y = x \times (1/y)$ のように分解する。乗算については既に述べたので、ここでは $1/x$ の計算方法について述べる。

$x = x_0 + \boldsymbol{x}^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k + \delta_x \varepsilon_{rx}$ の値域を $[a, b]$ とし、$1/x$ の $[a, b]$ での近似多項式を $rx^2 + px + q$ とする。$E \coloneqq \max_{x \in [a,b]} \left| \frac{1}{x} - (rx^2 + px + q) \right|$ とおくと、$1/x$ は

$$\begin{aligned}
1/x &\in rx^2 + px + q + E \cdot [-1, 1] \\
&= \underbrace{(rx_0^2 + px_0 + q)}_{=:z_0} + \underbrace{(2rx_0 + p)\boldsymbol{x}^T}_{=:\boldsymbol{z}}\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T \underbrace{(r\boldsymbol{x}\boldsymbol{x}^T + (2rx_0 + p)X)}_{=:Z}\boldsymbol{\varepsilon}_k \\
&\quad + r\boldsymbol{\varepsilon}_k^T \underbrace{(X\boldsymbol{\varepsilon}_k\boldsymbol{\varepsilon}_k^T X + 2X\boldsymbol{\varepsilon}_k\boldsymbol{x}^T)}_{\in [A]\text{：区間行列}}\boldsymbol{\varepsilon}_k + (rx + p)\delta_x\varepsilon_{rx} + E \cdot [-1, 1] \\
&= z_0 + \boldsymbol{z}^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T \underbrace{(Z + r\,\mathrm{mid}([A]))}_{=:Z'}\boldsymbol{\varepsilon}_k + \underbrace{r\boldsymbol{\varepsilon}_k^T([A] - \mathrm{mid}([A]))\boldsymbol{\varepsilon}_k + (rx + p)\delta_x\varepsilon_{rx} + E \cdot [-1, 1]}_{\text{ここを評価して、結果を } E' \text{ とおく}} \\
&\subset z_0 + \boldsymbol{z}^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T Z' \boldsymbol{\varepsilon}_k + \mathrm{rad}(E')[-1, 1]
\end{aligned}$$

と計算できる。E' は、乗算と同様に $[A]$, $[A] - \mathrm{mid}([A])$ を評価して、x を (7), (8) を用いて評価することで計算できる。最後の項を新しいダミー変数 ε_{k+1} と ε_{rz} を用いて表すと、$1/x$ は

$$z_0 + \begin{pmatrix} \boldsymbol{z} \\ \mathrm{rad}(E') \end{pmatrix}^T \boldsymbol{\varepsilon}_{k+1} + \boldsymbol{\varepsilon}_{k+1}^T \begin{pmatrix} \frac{Z' + Z'^T}{2} & \begin{matrix} 0 \\ \vdots \end{matrix} \\ \begin{matrix} 0 & \cdots \end{matrix} & 0 \end{pmatrix} \boldsymbol{\varepsilon}_{k+1} + 0\varepsilon_{rz}$$

と計算できる。各演算で発生する丸め誤差は、乗算と同様に ε_{k+1} の係数に加える。近似多項式の求め方については 5 章で述べる。

5　拡張 affine 演算における $1/x$ の計算

拡張 affine 演算では、除算の説明で述べたように、$1/x$ を近似する多項式 rx^2+px+q を用いて次のように $1/x$ を計算する。

$$\begin{aligned} &E \coloneqq \max_{x\in[a,b]} \left|\frac{1}{x} - (rx^2+px+q)\right|, \\ &\frac{1}{x} \in rx^2+px+q+E\cdot[-1,1]. \end{aligned} \tag{9}$$

ここで、$[a,b]$ は $x = x_0 + \boldsymbol{x}^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k + \delta_x\varepsilon_{rx}$ の値域である。以下では、$a>0$ と仮定して、(9) の求め方について述べる。

5.1　方法 1: Chebyshev 補間

5.1.1　方法 1-1: 方程式を解いて求める

区間 $[a,b]$ 上での Chebyshev 補間を求めるには、補間点として (3) を用いれば良い。二次の多項式を求めたいので、

$$x_i = \cos\left(\frac{\pi\left(i-\frac{1}{2}\right)}{n}\right)\frac{b-a}{2}+\frac{a+b}{2} \quad (i=1,\ 2,\ 3) \tag{10}$$

の三点を補間点とする。次の連立一次方程式を解くことで (9) の $r,\ p,\ q$ を求めることができる。

$$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}\begin{pmatrix} r \\ p \\ q \end{pmatrix} = \begin{pmatrix} 1/x_1 \\ 1/x_2 \\ 1/x_3 \end{pmatrix}. \tag{11}$$

この計算は通常の浮動小数点演算で行えば良い。

5.1.2　方法 1-2: 方程式を解かずに求める

(11) を解く代わりに、(4), (5) を使うことでも Chebyshev 補間を計算できる。(4) で $k=3$ として $c_0,\ c_1,\ c_2$ を求めて、$s \coloneqq (a+b)/2,\ t \coloneqq (b-a)/2$ とおくと、

$$\begin{aligned} \left(\sum_{j=0}^{2} c_j T_j\left(\frac{x-s}{t}\right)\right) - \frac{1}{2}c_0 &= \left(2\left(\frac{x-s}{t}\right)^2-1\right)c_2 + \left(\frac{x-s}{t}\right)c_1 + c_0 - \frac{1}{2}c_0 \\ &= \left(2\left(\frac{x^2}{t^2}-2\frac{s}{t^2}x+\frac{s^2}{t^2}\right)-1\right)c_2 + \left(\frac{x}{t}-\frac{s}{t}\right)c_1 + c_0 - \frac{1}{2}c_0 \\ &= \frac{2c_2}{t^2}x^2 + \left(\frac{1}{t}c_1 - \frac{4s}{t^2}c_2\right)x + \left(\frac{2s^2}{t^2}-1\right)c_2 - \frac{s}{t}c_1 + \frac{1}{2}c_0 \end{aligned}$$

となるので、

$$r = \frac{2c_2}{t^2},\ p = \left(\frac{1}{t}c_1 - \frac{4s}{t^2}c_2\right),\ q = \left(\frac{2s^2}{t^2}-1\right)c_2 - \frac{s}{t}c_1 + \frac{1}{2}c_0$$

となる。この計算は通常の浮動小数点演算で行えば良い。

5.1.3 誤差関数の極値の計算

誤差関数 $e(x) = \frac{1}{x} - (rx^2 + px + q)$ の極値の位置がわかっていると、$E = \max_{x\in[a,b]} \left|\frac{1}{x} - (rx^2 + px + q)\right|$ を効率よく求めることができる。誤差関数を微分すると

$$e'(x) = \frac{-1 - 2rx^3 - px^2}{x^2}$$

なので、

$$2rx^3 + px^2 + 1 = 0 \tag{12}$$

を満たす x に対して $e'(x) = 0$ となる。$y \coloneqq \frac{1}{x}$ とおくと、(12) は

$$f(y) \coloneqq y^3 + py + 2r = 0 \tag{13}$$

となる。(12) の解のうち $[a,b]$ に含まれるものが極値の位置となるので、(13) の解のうち $[1/b, 1/a]$ に含まれるものを求めれば、極値の位置がわかる。

$P(x)$ を、(10) の x_1, x_2, x_3 で $e(x) = 0$ となるように決めているので、Rolle の定理より二つの区間 (x_1,x_2), (x_2,x_3) それぞれに (12) の解が一つ以上存在する。よって、$[1/b, 1/a]$ 内の二つ以上の異なる点が (13) の解となる。また、(13) の解を y_1, y_2, y_3 とおいて、(13) の $f(y)$ を因数分解すると、

$$\begin{aligned} f(y) &= (y - y_1)(y - y_2)(y - y_3) \\ &= y^3 - (y_1 + y_2 + y_3)y^2 + (y_1y_2 + y_2y_3 + y_3y_1)y - y_1y_2y_3 \\ &= 0 \end{aligned}$$

で、y^2 の係数を (13) と比較すれば

$$y_1 + y_2 + y_3 = 0$$

となる。二つ以上の解が $[1/b, 1/a]$ にあることがわかっているので、$[1/b, 1/a]$ 内の異なる二つの点と、一つの負の実数が (13) の解となることがわかる。そこで、三つの解に対して $y_1 < 0 < y_2 < y_3$ という関係が成り立つとする。すると、$f(y)$ のグラフは図 3 のように描ける。$f(y)$ の極値の一つが $[1/b, 1/a]$ にあることがわかっているので、方程式

$$f'(y) = 3y^2 + p = 0.$$

は二つの実数解を持つ。よって、

$$p < 0$$

でなければならない。$f'(y) = 0$ の解は $y = \pm\sqrt{\frac{-p}{3}}$ で、図 3 のように、$f\left(\sqrt{\frac{-p}{3}}\right)$ と $f\left(-\sqrt{\frac{-p}{3}}\right)$ の符号が逆にならなければならないことがわかる。よって、

$$\begin{aligned} f\left(\sqrt{\frac{-p}{3}}\right) \cdot f\left(-\sqrt{\frac{-p}{3}}\right) &= \left(\sqrt{\frac{-p}{3}}^3 + p\sqrt{\frac{-p}{3}} + 2r\right)\left(-\sqrt{\frac{-p}{3}}^3 - p\sqrt{\frac{-p}{3}} + 2r\right) \\ &= \left(2\sqrt{\frac{-p}{3}}^3 + 2r\right)\left(-2\sqrt{\frac{-p}{3}}^3 + 2r\right) \\ &= -4\left(\frac{-p}{3}\right)^3 + 4r^2 \\ &< 0 \end{aligned} \tag{14}$$

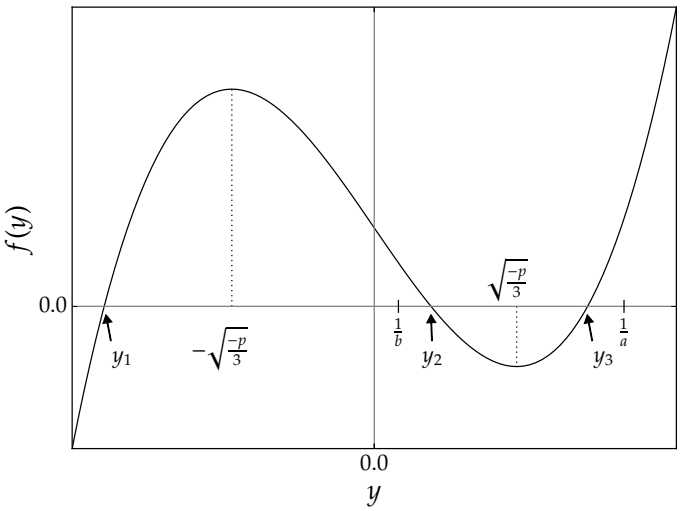


図 3: $f(y)$ の概形

が得られる。

ここでは、(13) を解くために三次方程式の代数的解法である Cardano の方法を使う。まず、

$$y = u + v$$

とおくと、(13) は

$$u^3 + v^3 + 2r + (3uv + p)(u + v) = 0$$

となる。よって、

$$\begin{aligned} u^3 + v^3 + 2r &= 0, \\ 3uv + p &= 0 \end{aligned}$$

を満たす u, v を探せば、(13) の解が求まる。この式から v を消去すると、

$$u^6 + 2ru^3 - \left(\frac{p}{3}\right)^3 = 0.$$

これを u^3 の二次方程式と見なして解くと、

$$u^3 = -r \pm \sqrt{r^2 + \left(\frac{p}{3}\right)^3}$$

が得られる。u^3 をこの二つの解のどちらかに選ぶと、もう一方の解が v^3 となる。$y = u + v$ なので、(13) の解は

$$\begin{aligned} y &= u + v \\ &= \sqrt[3]{-r + \sqrt{r^2 + \left(\frac{p}{3}\right)^3}} + \sqrt[3]{-r - \sqrt{r^2 + \left(\frac{p}{3}\right)^3}} \end{aligned} \tag{15}$$

を用いて計算できる。(14) から

$$r^2 + \left(\frac{p}{3}\right)^3 < 0$$

が得られるので、(15) は複素数の和として表せる。

$$\begin{aligned}\theta &\coloneqq \sqrt{-\left(r^2+\left(\frac{p}{3}\right)^3\right)},\\ \varphi &\coloneqq \arctan\left(\frac{\theta}{-r}\right)\end{aligned} \tag{16}$$

とおくと、

$$\begin{aligned}-r+i\theta &= |-r+i\theta|\,e^{i\varphi} = \sqrt{r^2+\theta^2}\,e^{i\varphi} = \sqrt{\left(-\frac{p}{3}\right)^3}\,e^{i\varphi},\\ -r-i\theta &= |-r-i\theta|\,e^{-i\varphi} = \sqrt{r^2+\theta^2}\,e^{-i\varphi} = \sqrt{\left(-\frac{p}{3}\right)^3}\,e^{-i\varphi}\end{aligned}$$

となるので、これを使うと、(15) は次のように書き直せる。

$$\begin{aligned}y &= \sqrt[3]{-r+i\theta}+\sqrt[3]{-r-i\theta}\\ &= \sqrt{-\frac{p}{3}}\,e^{i\left(\frac{\varphi}{3}+\frac{2}{3}k\pi\right)}+\sqrt{-\frac{p}{3}}\,e^{-i\left(\frac{\varphi}{3}+\frac{2}{3}k'\pi\right)}\quad (k,k'=0,\,1,\,2).\end{aligned}$$

よって、

$$y=\sqrt{-\frac{p}{3}}\left(\cos\left(\frac{\varphi}{3}+\frac{2}{3}k\pi\right)+\cos\left(-\frac{\varphi}{3}-\frac{2}{3}k'\pi\right)+i\sin\left(\frac{\varphi}{3}+\frac{2}{3}k\pi\right)+i\sin\left(-\frac{\varphi}{3}-\frac{2}{3}k'\pi\right)\right)\quad (k,k'=0,\,1,\,2)$$

となり、(13) の解は三つの実数であることがわかっているので、

$$\sin\left(\frac{\varphi}{3}+\frac{2}{3}k\pi\right)+\sin\left(-\frac{\varphi}{3}-\frac{2}{3}k'\pi\right)=\sin\left(\frac{\varphi}{3}+\frac{2}{3}k\pi\right)-\sin\left(\frac{\varphi}{3}+\frac{2}{3}k'\pi\right)=0$$

となる k, k' の組み合わせから (13) の解が得られる。n を任意の整数とすると、

$$\frac{\varphi}{3}+\frac{2}{3}k\pi=\frac{\varphi}{3}+\frac{2}{3}k'\pi+2n\pi$$

なので、

$$k=k'+3n$$

となる。$0\le k,k'\le 2$ なので、

$$k=k'=0,\ k=k'=1,\ k=k'=2$$

の三つの組み合わせが得られ、(13) の解は

$$y=2\sqrt{-\frac{p}{3}}\cos\left(\frac{\varphi+2k\pi}{3}\right)\quad (k=0,\,1,\,2) \tag{17}$$

となる。これらの解のうち $[1/b,1/a]$ に含まれる二つの解 y_2, y_3 が極値の座標を与える。

$$x_0\coloneqq a,\ x_1\coloneqq b,\ x_2\coloneqq\frac{1}{y_2},\ x_3\coloneqq\frac{1}{y_3}$$

とおくと、(9) の E を

$$E=\max_i\left|\frac{1}{x_i}-(rx_i^2+px_i+q)\right| \tag{18}$$

と求めることができる。

(18) の計算を x_i の算定も含めて区間演算などを用いて精度保証付きで行えば、(9) を用いて拡張 affine 演算における $1/x$ を計算できる。

5.2 方法 2: 最良近似

Remes の第二アルゴリズムを基にすると、次のアルゴリズムで最良近似を求めることができる。ただし、入力として近似の範囲 $[a, b]$、反復回数の上限 $k_{\max}$、許容誤差 E_{tol} が与えられているとする。

1. Chebyshev 補間などを用いて、区間 (a, b) 内の三点で $1/x$ と一致するように近似多項式 $r_0x^2+p_0x+q_0$ のパラメータ r_0, p_0, q_0 を求める。
2. 誤差関数 $e_0(x) = \frac{1}{x} - (r_0x^2 + p_0x + q_0)$ の極値の座標 $x_{0,1}$, $x_{0,2}$ を、$x_{0,1} < x_{0,2}$ となるように求める。また、$x_{0,0} \coloneqq a$, $x_{0,3} \coloneqq b$ とおき、
$$E_0 \coloneqq \max_i \left|e_0(x_{0,i})\right|$$
とおく。
3. $k \coloneqq 1$ とおく。
4. 次の連立方程式を解く。
$$\begin{pmatrix} x_{k-1,0}^2 & x_{k-1,0} & 1 & 1 \\ x_{k-1,1}^2 & x_{k-1,1} & 1 & -1 \\ x_{k-1,2}^2 & x_{k-1,2} & 1 & 1 \\ x_{k-1,3}^2 & x_{k-1,3} & 1 & -1 \end{pmatrix} \begin{pmatrix} r_k \\ p_k \\ q_k \\ E'_k \end{pmatrix} = \begin{pmatrix} \frac{1}{x_{k-1,0}} \\ \frac{1}{x_{k-1,1}} \\ \frac{1}{x_{k-1,2}} \\ \frac{1}{x_{k-1,3}} \end{pmatrix}$$
5. 誤差関数 $e_k(x) = \frac{1}{x} - (r_kx^2 + p_kx + q_k)$ の極値の座標 $x_{k,1}$, $x_{k,2}$ を、$x_{k,1} < x_{k,2}$ となるように求める。また、$x_{k,0} \coloneqq a$, $x_{k,3} \coloneqq b$ とおき、
$$E_k \coloneqq \max_i \left|e_k(x_i)\right|$$
とおく。
   - 中間値の定理より区間 $(x_{k-1,0}, x_{k-1,1})$, $(x_{k-1,1}, x_{k-1,2})$, $(x_{k-1,2}, x_{k-1,3})$ それぞれの中に $e_k(x) = 0$ となる点が存在するので、5.1.3 節の (17) を利用して $x_{k,1}$, $x_{k,2}$ を求めることができる。
6. もし $k = k_{\max}$ または $|E_k| - |E_{k-1}| < E_{\text{tol}}$ なら終了する。そうでなければ、$k \coloneqq k + 1$ とおきなおして 4. からの手順を繰り返す。

上のアルゴリズムの各ステップでの計算は浮動小数点数による近似計算で問題ない。アルゴリズムの出力として、近似多項式 $r_kx^2 + p_kx + q_k$ が得られる。この多項式を使って、E の計算を、極値の座標の算定も含めて区間演算などを用いて精度保証付きで行うと、(9) を用いて拡張 affine 演算における $1/x$ を計算できる。E は (18) を使って計算できる。

5.3 方法 3: 別の方法

$1/x$ に対して
$$x \times (1/x) = 1$$
となることを利用して、多項式で表される次の誤差関数
$$f(x) \coloneqq x(rx^2 + px + q) - 1 \tag{19}$$

を考える。$x = x_0 + \Delta x$ とおいて、$f(x)$ を x_0 の周りで Taylor 展開すると次のようになる。

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{3!}f'''(x_0)(x - x_0)^3 \\ &= (rx_0^3 + px_0^2 + qx_0 - 1) + (3rx_0^2 + 2px_0 + q)\Delta x + (3rx_0 + p)\Delta x^2 + r\Delta x^3. \end{aligned} \tag{20}$$

このとき、(20) の Δx の 0 次、1 次、2 次の項が消えるように係数を決める方法が方法 3 である。(6) と比較すると $\Delta x = \boldsymbol{x}^T\boldsymbol{\varepsilon}_k + \boldsymbol{\varepsilon}_k^T X \boldsymbol{\varepsilon}_k + \delta_x \varepsilon_{rx}$ となるので、(19) の形の誤差関数を考えたときにダミー変数の高次項だけが残るように係数を決める方法、と言い換えることもできる。Δx の 0 次、1 次、2 次の項を消すためには連立方程式

$$\begin{cases} rx_0^3 + px_0^2 + qx_0 - 1 = 0 \\ 3rx_0^2 + 2px_0 + q = 0 \\ 3rx_0 + p = 0 \end{cases}$$

を解けば良いので、これを解くと

$$r = \frac{1}{x_0^3},\ p = -\frac{3}{x_0^2},\ q = \frac{3}{x_0}$$

が得られる。$r,\ p,\ q$ を上の式で浮動小数点演算で計算したものを $r',\ p',\ q'$ とおき、丸め誤差による真値からのずれを

$$r' = r + \Delta r,\ p' = p + \Delta p,\ q' = q + \Delta q$$

と表現する。このとき、誤差関数 $e(x) = \frac{1}{x} - (r'x^2 + p'x + q')$ の導関数は、

$$\begin{aligned} e'(x) &= -\frac{1}{x^2} - 2r'x - p' \\ &= -\frac{1}{x^2} - 2(r + \Delta r)x - (p + \Delta p) \\ &= -\frac{1}{x^2}(1 + 2rx^3 + px^2) - (2\Delta r\, x + \Delta p) \\ &= -\frac{1}{x^2}\left(\frac{2}{x_0^3}x^3 - \frac{3}{x_0^2}x^2 + 1\right) - (2\Delta r\, x + \Delta p) \\ &= -\frac{1}{x^2}(x - x_0)^2\left(\frac{2}{x_0^3}x + \frac{1}{x_0^2}\right) - (2\Delta r\, x + \Delta p) \end{aligned}$$

となる。$a > 0$ より $x > 0,\ x_0 > 0$ なので、

$$-\frac{1}{x^2}(x - x_0)^2\left(\frac{2}{x_0^3}x + \frac{1}{x_0^2}\right) \le 0$$

が成り立つ。また、$r',\ p'$ を上向き丸めで計算していれば

$$\Delta r, \Delta p \ge 0$$

となるので、任意の $x > 0$ に対して

$$-(2\Delta r\, x + \Delta p) \le 0$$

となる。よって、$r',\ p'$ を上向き丸めで計算していれば $e'(x) \le 0$ で、$e(x)$ は単調減少となる。なので、方法 3 では (9) の E は

$$E = \max\left\{\left|\frac{1}{a} - (r'a^2 + p'a + q')\right|, \left|\frac{1}{b} - (r'b^2 + p'b + q')\right|\right\}$$

のように値域の両端での誤差を計算すれば求められる。

5.3.1 他の演算への拡張

$1/x$ についての数値実験 (7 章と 8 章) では、方法 1 や方法 2 は区間幅が狭い場合などに計算に失敗する場合がみられる。このことから、平方根などの演算についても方法 3 と同様な計算が行える必要があると考えられる。7 章と 8 章の数値実験では扱わないが、方法 3 と同様な計算が他の演算に対しても行えるということを示すために、平方根の係数と近似の誤差の計算方法を述べ、さらに三角関数への拡張の方法の方針を記しておく。

平方根については、適当な定義域 $[a, b]$ で $\sqrt{x} \approx rx^2 + px + q$ となるように r, p, q を決めたいということと、

$$\sqrt{x}^2 = x$$

を用いることで (19) と同様に多項式による誤差関数を構成することを考えることができ、

$$f(x) \coloneqq (rx^2 + px + q)^2 - x$$

となる。$x = x_0 + \Delta x$ とおいて、この関数を x_0 の周りで Taylor 展開すると、

$$\begin{aligned}
f(x) &= (r^2x_0^4 + 2rpx_0^3 + (2rq + p^2)x_0^2 + (2pq - 1)x_0 + q^2) \\
&\quad + (4r^2x_0^3 + 6rpx_0^2 + 2(2rq + p^2)x_0 + (2pq - 1))\Delta x \\
&\quad + \frac{1}{2}(12r^2x_0^2 + 12rpx_0 + 2(2rq + p^2))\Delta x^2 \\
&\quad + \frac{1}{3!}(24r^2x_0 + 12rp)\Delta x^3 + \frac{1}{4!}24r^2\Delta x^4 \\
&= (r^2x_0^4 + 2rpx_0^3 + (2rq + p^2)x_0^2 + (2pq - 1)x_0 + q^2) \\
&\quad + (4r^2x_0^3 + 6rpx_0^2 + 2(2rq + p^2)x_0 + (2pq - 1))\Delta x \\
&\quad + (6r^2x_0^2 + 6rpx_0 + (2rq + p^2))\Delta x^2 \\
&\quad + (4r^2x_0 + 2rp)\Delta x^3 + r^2\Delta x^4
\end{aligned}$$

となるので、Δx の 0 次、1 次、2 次の項を消すために解く方程式は

$$\begin{cases}
r^2x_0^4 + 2rpx_0^3 + (2rq + p^2)x_0^2 + (2pq - 1)x_0 + q^2 = 0 \\
4r^2x_0^3 + 6rpx_0^2 + 2(2rq + p^2)x_0 + (2pq - 1) = 0 \\
6r^2x_0^2 + 6rpx_0 + (2rq + p^2) = 0
\end{cases}$$

となる。この方程式の解を求めると、

$$r = -\frac{1}{8\sqrt{x_0}^3},\ p = \frac{3}{4\sqrt{x_0}},\ q = \frac{3\sqrt{x_0}}{8}$$

が得られる。r, p, q を上の式で浮動小数点演算で計算したものを r', p', q' とおき、丸め誤差による真値からのずれを

$$r' = r + \Delta r,\ p' = p + \Delta p,\ q' = q + \Delta q$$

と表現する。このとき、誤差関数 $e(x) = \sqrt{x} - (r'x^2 + p'x + q')$ の導関数は

$$\begin{aligned}
e'(x) &= \frac{1}{2\sqrt{x}} - (2r'x + p') \\
&= \frac{1}{2\sqrt{x}} - (2(r + \Delta r)x + (p + \Delta p)) \\
&= x\left(\frac{1}{2\sqrt{x}^3} - 2r - \frac{p}{\sqrt{x}^2}\right) - (2\Delta r\, x + \Delta p) \\
&= x\left(\frac{1}{2\sqrt{x}^3} - 2\left(-\frac{1}{8\sqrt{x_0}^3}\right) - \frac{3}{4\sqrt{x_0}}\frac{1}{\sqrt{x}^2}\right) - (2\Delta r\, x + \Delta p) \\
&= \frac{x}{2}\left(\frac{1}{\sqrt{x}} - \frac{1}{\sqrt{x_0}}\right)^2\left(\frac{1}{\sqrt{x}} + \frac{1}{2\sqrt{x_0}}\right) - (2\Delta r\, x + \Delta p)
\end{aligned}$$

となる。$x, x_0 \geq 0$ でなければならないので

$$\frac{x}{2}\left(\frac{1}{\sqrt{x}} - \frac{1}{\sqrt{x_0}}\right)^2\left(\frac{1}{\sqrt{x}} + \frac{1}{2\sqrt{x_0}}\right) \geq 0$$

で、r', p' を下向き丸めで計算していれば

$$\Delta r, \Delta p \leq 0$$

となるので

$$-(2\Delta r\, x + \Delta p) \geq 0$$

となる。よって、r', p' を下向き丸めで計算していれば常に $e'(x) \geq 0$ で、$e(x)$ は単調増加となることがわかる。なので、近似の範囲 $[a, b]$ の両端での誤差から近似の誤差を求めることができる。

三角関数については、適当な定義域で $\sin(x) \approx rx^2 + px + q$ となるように r, p, q を決めたいということと、

$$\sin^2(x) + \cos^2(x) = \sin^2(x) + \sin^2\left(x + \frac{\pi}{2}\right) = 1$$

を使えば、(19) と同様に多項式による誤差関数が構成できるので、その関数を x_0 の周りで Taylor 展開して Δx の 0 次、1 次、2 次の項が消えるように係数 r, p, q を決めることで、方法 3 と同様な計算ができると考えられる。

6 作成したライブラリ

ここまでに説明した手法のうち、5.3.1 節に述べたもの以外を実装して、qifen [*1] という名前のライブラリを作成した。`https://bitbucket.org/uec-dn/qifen` からダウンロードすることができる。Git というツールを使ってダウンロードすると、最新版への更新などが簡単にできる。

ソースファイル一式をダウンロードし、`build` ディレクトリ内で

```
cmake -G "Unix Makefiles" ..
make
```

[*1] 「Quadratic form を使った Interval arithmetic」を略して qi。漆 (中国語での読みが qī [ʧʰi˥]) の主成分 (ある意味で「漆の素」のようなもの) がウルシオール (中国語で漆酚、読みは qīfēn [ʧʰi˥.fən˥])。二次形式を使った区間演算 (qi) の基になるライブラリなので qifen。

を実行するとライブラリとサンプルプログラムが生成される。Windows では、`cmake -G "Visual Studio 15 2017 Win64" ..` のようなコマンドを実行して、生成される `sln` ファイルを Visual Studio で開けば良い。コンパイルには Boost [10] が必要である。言語は C++ で書かれているが、生成されたライブラリは C 言語からでも利用することができる。このライブラリは、内部で C++ で書かれた精度保証付き数値計算ライブラリである kv ライブラリ [11] や MATLAB や Octave 上で精度保証付き数値計算を行うためのパッケージである INTLAB [12] を利用することで区間演算を実装している。

次の構造体が定義されている。

- `qifen_interval_t`: 区間を表す
- `qifen_matrix_t`: 行列を表す
- `qifen_interval_matrix_t`: 区間行列を表す
- `qifen_qi_context_t`: ダミー変数の数など、拡張 affine 演算に関する情報を表す
- `qifen_qi_t`: 拡張 affine 演算に用いる変数を表す

6.1 近似方法の選択

非線形単項演算の近似方法を指定するための列挙型 `qifen_approx_method_t` が用意されており、以下の値が定義されている。

- `qifen_approx_best_effort`: 最良近似、Chebyshev 補間、方法 3(もしくは一次近似) の順で試す
- `qifen_approx_linear`: affine 演算と同様に一次式で近似する
- `qifen_approx_chebyshev`: (11) を解いて Chebyshev 補間を求める
- `qifen_approx_chebyshev_2`: Chebyshev 補間を使うが、方程式を解かずに求める
- `qifen_approx_remez`: 最良近似を求める
- `qifen_approx_fast`: 方法 3 を使う

割り算などの非線形な単項演算を行う関数はすべて引数に `qifen_approx_method_t` を含むので、この値を変えることで近似の方法を選択することができる。

6.2 計算の成否

値域に 0 が含まれる x に対して $1/x$ を計算しようとしたときなどに計算に失敗することがあるので、計算の成否を表す列挙型 `qifen_error_t` が用意されている。定義されている値は以下の通り。

- `qifen_error_succeeded`: 計算に成功したことを表す
- `qifen_error_domain`: 入力の値域に問題があることを表す
- `qifen_error_eig`: 固有値の精度保証に失敗したことを表す
- `qifen_error_matrix_size`: 入力の行列の大きさに問題があることを表す
- `qifen_error_approx_failed`: 非線形演算の近似多項式が求められなかったことを表す

失敗する可能性のある関数はこれらのうちいずれかの値を返す。

6.3 ライブラリの設定

qifen_interval_t と qifen_interval_matrix_t は内部で kv か INTLAB を利用して区間値を保持する。どちらを利用するかは、ライブラリをコンパイルする時と使用する時に次のどちらかのマクロを定義することで選ぶことができる。

- QIFEN_CONFIG_INTERVAL_KV: kv を利用する
- QIFEN_CONFIG_INTERVAL_INTLAB: INTLAB を利用する

また、ライブラリをコンパイルする時と使用する時に次のマクロを定義することで、不要な関数を無効化する (ライブラリに含まれないようにする) ことができる。

- QIFEN_CONFIG_DISABLE_KV: kv に関係する関数を無効化する
- QIFEN_CONFIG_DISABLE_INTLAB: INTLAB に関係する関数を無効化する

以下に、区間値を保持するのに kv を利用する場合と INTLAB を利用する場合のプログラムの例を載せる。

プログラム 1: config-kv.cpp

```
1 // 区間値の保持に kv を利用して INTLAB に関する関数は無効化する
2 #define QIFEN_CONFIG_INTERVAL_KV
3 #define QIFEN_CONFIG_DISABLE_INTLAB
4
5 #include <qifen.h>
6
7 int main()
8 {
9 }
```

プログラム 2: config-intlab.cpp

```
1 // 区間値の保持に INTLAB を利用して kv に関する関数は無効化する
2 #define QIFEN_CONFIG_INTERVAL_INTLAB
3 #define QIFEN_CONFIG_DISABLE_KV
4
5 #include <qifen.h>
6
7 int main()
8 {
9 }
```

6.4 初期化・後処理を行う関数

すべての構造体に対して、初期化と後処理を行う関数が用意されている。変数を使う前に初期化用の関数を呼び、変数が不要になったら後処理用の関数を呼ぶ必要がある。初期化用の関数は以下の通り。

- void qifen_interval_init(qifen_interval_t);
- void qifen_matrix_init(qifen_matrix_t);
- void qifen_interval_matrix_init(qifen_interval_matrix_t);
- void qifen_qi_context_init(qifen_qi_context_t);
- void qifen_qi_init(qifen_qi_context_t, qifen_qi_t);

これらの関数の名前を init を clear に変えたものが後処理用の関数として用意されている。

qifen_qi_t については、初期化と値の設定を同時に行う関数が三つ用意されている。

- void qifen_qi_init_interval(qifen_qi_context_t ctx, qifen_qi_t v, const qifen_interval_t a);
  - qifen_interval_t が表す区間を値域とするように初期化する
- void qifen_qi_init_infsup(qifen_qi_context_t ctx, qifen_qi_t v, double inf, double sup);
  - 区間 [inf,sup] を値域とするように初期化する
- void qifen_qi_init_infsup_string(qifen_qi_context_t ctx, qifen_qi_t v, const char *inf, const char *sup);
  - 文字列 inf を下向き丸めで浮動小数点数に変換したものを値域の下限とし、文字列 sup を上向き丸めで浮動小数点数に変換したものを値域の上限とするように初期化する

例えば、qifen_qi_t を使う場合は、次のように初期化と後処理を実行する必要がある。

プログラム 3: qi-init-clear.cpp

```
1  // 区間値の保持に kv を利用して INTLAB に関する関数は無効化する
2  #define QIFEN_CONFIG_INTERVAL_KV
3  #define QIFEN_CONFIG_DISABLE_INTLAB
4
5  #include <qifen.h>
6
7  int main()
8  {
9      ::qifen_qi_context_t ctx;
10     ::qifen_qi_t x;
11
12     // ctx を初期化してから x を初期化する
13     ::qifen_qi_context_init(ctx);
14     ::qifen_qi_init(ctx, x);
15
16     // ここで x を使った計算を行う
17
18     // x の後処理を実行してから ctx の後処理を実行する
19     ::qifen_qi_clear(ctx, x);
20     ::qifen_qi_context_clear(ctx);
21 }
```

6.5 値を取得・設定する関数

qifen_qi_t については、次の関数が値を取得・設定するための関数として用意されている。

- void qifen_qi_set(qifen_qi_context_t ctx, qifen_qi_t dest, const qifen_qi_t v);
- void qifen_qi_set_infsup(qifen_qi_context_t ctx, qifen_qi_t v, double inf, double sup);
- void qifen_qi_set_infsup_string(qifen_qi_context_t ctx, qifen_qi_t v, const char *inf, const char *sup);
- void qifen_qi_get_infsup(qifen_qi_context_t ctx, qifen_qi_t v, double *inf, double *sup, qifen_verify_eig_method_t method = qifen_verify_eig_auto);

他の構造体については省略する。

6.6 演算を行う関数

qifen_qi_t に演算を行うための関数を以下に挙げる。

- void qifen_qi_negate(qifen_qi_context_t ctx, qifen_qi_t dest, const qifen_qi_t a);
  - dest $= -$a を計算する
- void qifen_qi_add_scalar(qifen_qi_context_t ctx, qifen_qi_t dest, const qifen_qi_t a, double b);
- void qifen_qi_add(qifen_qi_context_t ctx, qifen_qi_t dest, qifen_qi_t a, qifen_qi_t b);
  - dest $=$ a $+$ b を計算する
- void qifen_qi_sub(qifen_qi_context_t ctx, qifen_qi_t dest, qifen_qi_t a, qifen_qi_t b);
  - dest $=$ a $-$ b を計算する
- void qifen_qi_mul_scalar(qifen_qi_context_t ctx, qifen_qi_t dest, qifen_qi_t a, double b);
- void qifen_qi_mul(qifen_qi_context_t ctx, qifen_qi_t dest, qifen_qi_t a, qifen_qi_t b, qifen_verify_eig_method_t = qifen_verify_eig_auto);
  - dest $=$ a $\cdot$ b を計算する
- qifen_error_t qifen_qi_div(qifen_qi_context_t ctx, qifen_qi_t dest, qifen_qi_t a, qifen_qi_t b, const qifen_qi_inv_config_t config = qifen_qi_inv_config_default());
- qifen_error_t qifen_qi_div(qifen_qi_context_t ctx, qifen_qi_t dest, qifen_qi_t a, qifen_qi_t b, const qifen_qi_inv_config_struct &config);
  - dest $=$ a$/$b を計算する
- qifen_error_t qifen_qi_inv(qifen_qi_context_t ctx, qifen_qi_t dest, qifen_qi_t a, const qifen_qi_inv_config_t config = qifen_qi_inv_config_default());
- qifen_error_t qifen_qi_inv(qifen_qi_context_t ctx, qifen_qi_t dest, qifen_qi_t a, const qifen_qi_inv_config_struct &config);
  - dest $= 1/$a を計算する

qifen_qi_inv_config_t あるいは qifen_qi_inv_config_struct は、$1/x$ の近似の方法を指定するための構造体である。定義をプログラム 4 に載せる。各変数の意味は以下の通り。

- eig_method: 値域の評価での固有値の評価方法を表す
- approx_method: 近似多項式の計算方法を表す
- num_iter: 最良近似を求めるアルゴリズムの反復回数の上限 $k_{\max}$ を表す
- tolerance: 最良近似を求めるアルゴリズムの許容誤差 E_{tol} を表す

プログラム 4: qi-inv-config.cpp

```
typedef struct {
    qifen_verify_eig_method_t eig_method;
    qifen_approx_method_t approx_method;
    size_t num_iter;
    double tolerance;
} qifen_qi_inv_config_struct, qifen_qi_inv_config_t[1], *qifen_qi_inv_config_ptr;
```

6.7 プログラム例

拡張 affine 演算に用いる変数 x, y を、それぞれ $[1,2]$, $[3,4]$ を値域とするように初期化し、$z = xy/y$ を計算して z の値域を出力するプログラムと、その実行結果を示す。

プログラム 5: qifen-example.cpp

```
#define QIFEN_CONFIG_INTERVAL_KV
#define QIFEN_CONFIG_DISABLE_INTLAB
#include <qifen.h>

#include <iostream>

int main()
{
    ::qifen_qi_context_t ctx;
    ::qifen_qi_t x, y, z;

    ::qifen_qi_context_init(ctx);
    ::qifen_qi_init_infsup(ctx, x, 1.0, 2.0);
    ::qifen_qi_init_infsup(ctx, y, 3.0, 4.0);
    ::qifen_qi_init(ctx, z);

    ::qifen_qi_mul(ctx, z, x, y);
    ::qifen_qi_div(ctx, z, z, y, { ::qifen_approx_fast });

    double a, b;

    ::qifen_qi_get_infsup(ctx, z, &a, &b);

    ::std::cout << a << ' ' << b << ::std::endl;

    ::qifen_qi_clear(ctx, x);
    ::qifen_qi_clear(ctx, y);
    ::qifen_qi_clear(ctx, z);
    ::qifen_qi_context_clear(ctx);
}
```

プログラム 5 の実行結果

```
0.965136 2.03486
```

7 $1/x$ の近似方法の比較

6 章で述べたライブラリを用いて、5 章で述べた四つの $1/x$ の近似方法の比較を行う。用いた計算機は、CPU が Intel Core i7-4770、メモリが 32GB で、コンパイラは Visual C++ 2015 を用いた。

7.1 計算時間の比較

次の式を拡張 affine 演算で計算する。

$$\frac{1}{x},\ x \in [1.25, 2]. \tag{21}$$

拡張 affine 演算での逆数関数を 5 章で述べた四つの方法で計算することで、それぞれの方法の性能の違いを見る。計算して得られる値域と計算時間を表 1 に示す。

表 1: 逆数関数の近似方法の比較

	値域	区間幅	計算時間 [μs]
真の値域	$[0.5, 0.8]$	0.3	—
方法 1-1	$[0.46490458, 0.80000001]$	0.33509543	50897
方法 1-2	$[0.46490458, 0.80000001]$	0.33509543	50079
方法 2($k_{\max} = 1,\ E_{\text{tol}} = 0$)	$[0.46489181, 0.80001976]$	0.33512795	51320
方法 2($k_{\max} = 10,\ E_{\text{tol}} = 10^{-10}$)	$[0.46491106, 0.80000001]$	0.33508895	57752
方法 2($k_{\max} = 20,\ E_{\text{tol}} = 10^{-16}$)	$[0.46491106, 0.80000001]$	0.33508895	66048
方法 3	$[0.46354119, 0.80000001]$	0.33645882	7747

結果から、方法 3 は他の方法に比べて精度で劣ることがわかる。しかし、その差はあまり大きくなく、速度は方法 1 と比べると 6.5 倍以上高速なので、方法 3 が速度と精度のバランスが最も良いと考えられる。

7.2 入力を狭くした場合の比較

(21) の入力区間を次のように変更する。

$$x \in 1.25 + [-\varepsilon, \varepsilon].$$

区間半径 ε を小さくすると、方法 1-1、方法 1-2、方法 2 は計算に失敗した。それぞれの方法について、計算に成功した最小の区間半径 ε を表 2 に示す。

表 2: 計算に成功した最小の区間半径

	ε
方法 1-1	7.5×10^{-6}
方法 1-2	7.5×10^{-6}
方法 2	5.9574618×10^{-6}

いずれの方法も、極値の座標が区間 $[a, b]$ から外れることで計算に失敗する。これは、極値の計算をする際に解く方程式 (13) の解は、通常であれば図 3 のように区間 $[1/b, 1/a]$ の中に二つ存在するはずだが、失敗したときには図 4 のように解が $[1/b, 1/a]$ の外に存在するということを意味する。

方法 1-1 と方法 2 については、$\varepsilon = 7.5 \times 10^{-6}$ のときに (11) の左辺の行列の条件数がおよそ 3.5×10^{11} と大きくなることから、方程式を解く際の誤差が大きくなるためだと考えられる。方法 1-2 については、区間

$[a,b] = 1.25 + [-\varepsilon, \varepsilon]$ が狭くなると $\frac{b-a}{2} \approx 0$ となるので、(4) による c_2 の計算が、

$$
\begin{aligned}
c_2 &= \frac{2}{3}\sum_{j=1}^{3} f\left(\frac{b-a}{2}x_j + \frac{a+b}{2}\right) T_i(x_j) \\
&= \frac{2}{3}\sum_{j=1}^{3} f\left(\frac{b-a}{2}\cos\left(\frac{2j-1}{6}\pi\right) + \frac{a+b}{2}\right)\cos\left(\frac{2(2j-1)}{6}\pi\right) \\
&= \frac{2}{3}\left(f\left(\frac{b-a}{2}\cos\frac{1}{6}\pi + \frac{a+b}{2}\right)\cos\frac{\pi}{3} + f\left(\frac{b-a}{2}\cos\frac{1}{2}\pi + \frac{a+b}{2}\right)\cos\pi\right. \\
&\qquad \left. + f\left(\frac{b-a}{2}\cos\frac{5}{6}\pi + \frac{a+b}{2}\right)\cos\frac{5}{3}\pi\right) \\
&= \frac{2}{3}\left(f\left(\frac{b-a}{2}\frac{\sqrt{3}}{2} + \frac{a+b}{2}\right)\frac{1}{2} - f\left(\frac{a+b}{2}\right) + f\left(-\frac{b-a}{2}\frac{\sqrt{3}}{2} + \frac{a+b}{2}\right)\frac{1}{2}\right) \\
&\approx \frac{2}{3}\left(\frac{1}{2}f\left(\frac{a+b}{2}\right) + \frac{1}{2}f\left(\frac{a+b}{2}\right) - f\left(\frac{a+b}{2}\right)\right) \\
&= 0
\end{aligned}
$$

のようになり、桁落ちが発生するためだと考えられる。実際に $\varepsilon = 7.5 \times 10^{-6}$ のときの c_2 の計算を追うと

$$
\begin{aligned}
\frac{1}{2}\left(f\left(\frac{b-a}{2}\frac{\sqrt{3}}{2} + \frac{a+b}{2}\right) + f\left(-\frac{b-a}{2}\frac{\sqrt{3}}{2} + \frac{a+b}{2}\right)\right) &\approx 0.8000000000216 \\
f\left(\frac{a+b}{2}\right) &= 0.8 \\
c_2 &\approx 2.16 \times 10^{-11} \times \frac{2}{3} \\
&= 1.44 \times 10^{-11}
\end{aligned}
$$

となっており、桁落ちが発生していることが確認できる。

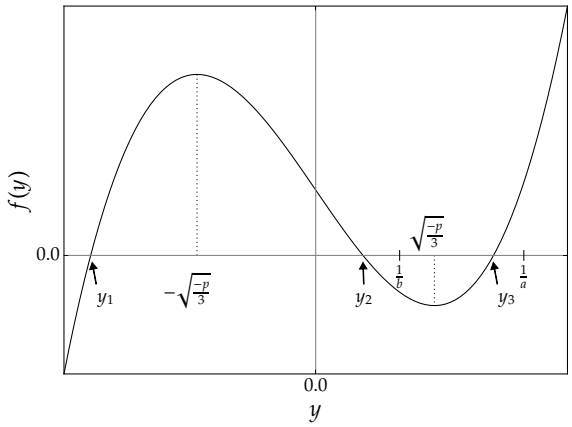


図 4: 計算に失敗した場合の (13) の $f(y)$ の概形

7.3　入力を変化させた場合の近似の誤差の比較

(21) の入力区間を次のように変更する。

$$x \in 10000 + [-\varepsilon, \varepsilon].$$

入力の区間半径 ε を変化させたときの、$1/x$ の近似の誤差 ((9) の E) をプロットすると、図 5 が得られる。

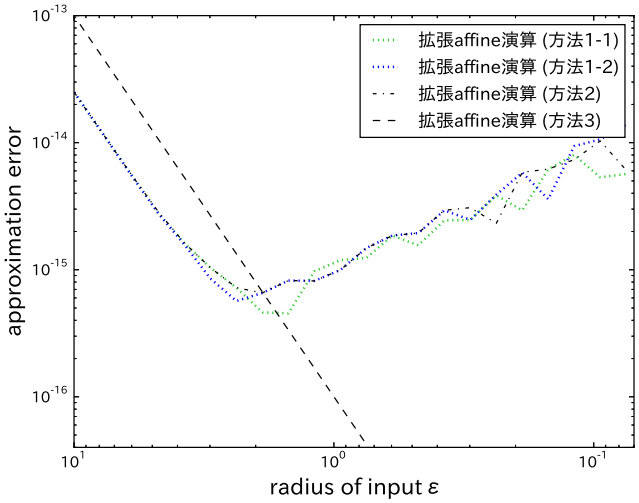


図 5: $x \in 10000 + [-\varepsilon, \varepsilon]$ に対する $1/x$ の近似の誤差

図を見ると方法 1 と方法 2 は誤差が一定の値以下にはならないことがわかるが、計算を追うと、原因が (16) にある割り算

$$\frac{\theta}{-r}$$

を区間演算で計算する際に区間幅が 10^{-11} 程度に拡大するためであることがわかる。また、$1/x$ の近似に対する誤差関数 $e(x) = 1/x - (rx^2 + px + q)$ の極値を求める際に解く方程式 (13) が $y > 0$ に重根を持つとすると、図 3 から $f\left(\sqrt{\frac{-p}{3}}\right) = 0$ となることがわかるので、

$$\begin{aligned} f\left(\sqrt{\frac{-p}{3}}\right) \cdot f\left(-\sqrt{\frac{-p}{3}}\right) &= \left(\sqrt{\frac{-p}{3}}^3 + p\sqrt{\frac{-p}{3}} + 2r\right)\left(-\sqrt{\frac{-p}{3}}^3 - p\sqrt{\frac{-p}{3}} + 2r\right) \\ &= \left(2\sqrt{\frac{-p}{3}}^3 + 2r\right)\left(-2\sqrt{\frac{-p}{3}}^3 + 2r\right) \\ &= -4\left(\frac{-p}{3}\right)^3 + 4r^2 \\ &= 0 \end{aligned}$$

となり、

$$r^2 + \left(\frac{p}{3}\right)^3 = 0$$

が得られる。近似の範囲が狭くなると誤差関数の極値が近くなるため、(13) の解は重根に近くなると考えられる。そのため

$$r^2 + \left(\frac{p}{3}\right)^3 \approx 0$$

となって、(16) の

$$\theta = \sqrt{-\left(r^2 + \left(\frac{p}{3}\right)^3\right)}$$

を区間演算で計算する際の区間拡大が大きくなるので、これが $\varepsilon = 10^0$ から $\varepsilon = 10^{-1}$ にかけて誤差が増加する原因だと考えられる。

8 affine 演算と拡張 affine 演算の比較

数値実験によって、拡張 affine 演算が有用な場合と、そうでない場合を見る。実験に用いた計算機は、CPU が Intel Core i7-4770、メモリが 32GB で、コンパイラは Visual C++ 2015 を用いた。また、affine 演算は kv [11]、拡張 affine 演算は 6 章で述べたライブラリを用いて計算した。

8.1 項の打ち消しが起こらない式

演算の過程で項が打ち消し合わないような次の式を用いて、affine 演算と拡張 affine 演算の性能の違いを見る。

$$\frac{xy}{z},\ x \in [1,2],\ y \in [3,4],\ z \in [5,6]. \tag{22}$$

この式を affine 演算と拡張 affine 演算で評価して得られる値域と計算時間を表 3 に示す。なお、表にある最良乗算は、[4] の最良乗算を指している。

表 3: (22) の評価結果

	値域	区間幅	計算時間 [μs]
真の値域	$[0.5, 1.6]$	1.1	—
affine 演算 ((1) による乗算)	$[0.31702895, 1.6000001]$	1.2829711	2.6
affine 演算 (最良乗算)	$[0.32120426, 1.6000001]$	1.2787959	11
拡張 affine 演算	$[0.25493989, 1.6541511]$	1.3992112	169

拡張 affine 演算は、二次項を計算する分だけ計算時間がかかる上、二次項の評価でもいくらか過大評価が起こるので、全く相関がない区間どうしの演算では affine 演算に速度的にも精度的にも劣ることがわかる。

8.2 乗除算の性能評価

次の式を約分せずに計算し、affine 演算と拡張 affine 演算の乗除算の性能の違いを見る。

$$\frac{x^3}{x^3},\ x \in [100, 110]. \tag{23}$$

この式を affine 演算と拡張 affine 演算で評価して得られる区間と計算時間を表 4 に示す。ただし、表にある「最良乗算」は [4] の最良乗算を指している。

また、入力区間 $[100, 110]$ を n 個に等分割したものを $x_1, x_2, \ldots, x_n$ とおいて、

$$\bigcup_{i=1}^{n} \frac{x_i^3}{x_i^3}$$

を包含する区間を affine 演算で計算することを考える。計算される区間の幅が (23) を拡張 affine 演算で分割なしで評価したときの区間幅以下となるように分割数 n を決めると、分割数と計算時間は表 5 のようになる。

表 4: (23) の評価結果

	区間	区間幅	計算時間 [μs]
affine 演算 ((1) による乗算)	$[0.97346887, 1.0493460]$	0.07587713	4
affine 演算 (最良乗算)	$[0.97496547, 1.0233538]$	0.04838833	20
拡張 affine 演算	$[0.99107201, 1.0087172]$	0.01764519	145

表 5: (23) の入力を分割したときの結果

	分割数	計算時間 [μs]
affine 演算 ((1) による乗算)	3	18
affine 演算 (最良乗算)	2	51

二次項を加えることで、演算の精度が改善されることがわかる。しかし速度は affine 演算に大きく劣り、この場合は、拡張 affine 演算と同品質の出力が必要であれば、入力を分割して affine 演算で計算する方が良いといえる。

8.3 多項式の値域の評価

[4] の数値例にある多項式

$$p(x) := 0.6x^5 + 37.5x^4 + 935x^3 + 11625x^2 + 72072x + 38.33,\ x \in [-15, -10] \tag{24}$$

を使って、affine 演算と拡張 affine 演算の性能の違いを見る。この式の真の値域と、この式を affine 演算と拡張 affine 演算で Horner 法を使って評価した場合、すなわち

$$((((0.6x + 37.5)x + 935)x + 11625)x + 72072)x + 38.33$$

を計算した場合に得られる値域と計算時間を表 6 に示す。表にある最良乗算は、[4] の最良乗算を指している。

また、入力区間 $[-15, -10]$ を n 個に等分割したものを $x_1, x_2, \ldots, x_n$ とおき、

$$\bigcup_{i=1}^{n} p(x_i)$$

を包含する区間を、affine 演算を用いて計算することを考える。計算される区間の幅が、拡張 affine 演算で分割なしで評価した場合の区間幅以下となるように分割数 n を決めるとすると、n と計算時間は表 7 のようになる。

表 6: (24) の評価結果

	値域	区間幅	計算時間 [μs]
真の値域	$[-178229, -178182]$	47	—
affine 演算 ((1) による乗算)	$[-177091.80, 797710.84]$	9974802.64	36
affine 演算 (最良乗算)	$[-61271.77, 797710.84]$	858982.61	78
拡張 affine 演算	$[-187604.17, -168806.67]$	18797.5	317

表 7: (24) の入力を分割したときの結果

	分割数	計算時間 [μs]
affine 演算 ((1) による乗算)	4	146
affine 演算 (最良乗算)	3	221

拡張 affine 演算を使うと出力の精度は改善される。これは、計算の過程で項が打ち消し合うことはないが、affine 演算では乗算の度に二次の項を評価して一つの区間にまとめてしまうのに対し、拡張 affine 演算では二次の項も保持するようにしたので、三次以上の項を評価するだけで済んだためだと考えられる。しかし、速度は affine 演算の方がはるかに高速で、入力を分割して affine 演算で評価した方が良いことがわかる。

8.4 入力が一変数の場合の性能評価

[3] にある次の例を用いて、affine 演算と拡張 affine 演算の性能の違いを見る。

$$\begin{aligned} f(x) &= x^2 - 2x, \\ g(x) &= x(x+1)\left(\frac{1}{x} - \frac{1}{x+1}\right), \\ &\text{evaluate } f(g(x)), \\ x &\in 10000 + [-\varepsilon, \varepsilon]. \end{aligned} \tag{25}$$

$g(x) = 1$ なので $f(g(x)) = -1$ だが、上の式の通り計算する。入力の区間半径と出力の区間半径との関係を図 6 に示す。affine 演算の乗算は [4] の最良乗算を用いている。

また、(25) で $\varepsilon = 1.0$ とし、入力区間 $10000 + [-\varepsilon, \varepsilon]$ を n 個に等分割したものを $x_1, x_2, \ldots, x_n$ とおく。そして、

$$\bigcup_{i=1}^{n} f(g(x_i))$$

を包含する区間を、affine 演算と拡張 affine 演算それぞれで計算する。計算される区間の幅が 10^{-15} 以下になるように分割数 n を決めるとすると、それぞれの演算について分割数と計算時間は表 8 のようになる。

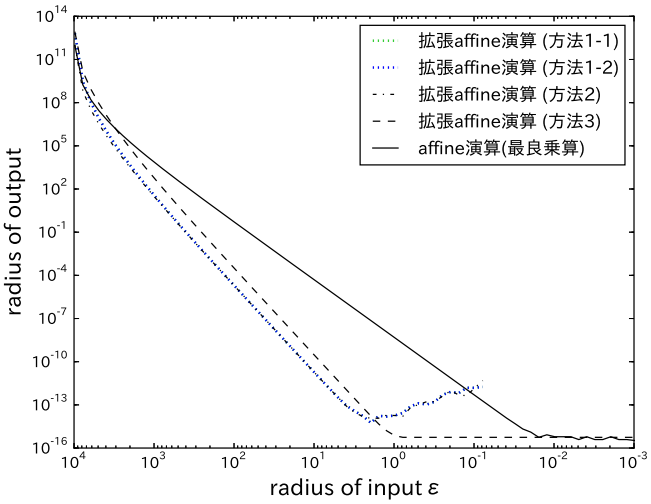


図 6: 入力が一変数の場合の性能評価の結果

表 8: 入力が一変数の場合の性能評価の結果 2

	分割数	計算時間 [μs]
affine 演算 ((1) による乗算)	63	323
affine 演算 (最良乗算)	58	3055
拡張 affine 演算 (方法 3)	1	265

結果から、affine 演算では、入力の分割による計算時間の増加が大きくなるため、affine 演算より拡張 affine 演算の方が少しだけ速度の面で有利であることがわかる。

方法 1 と方法 2 は、入力を小さくしているにもかかわらず出力が大きくなっている部分がある。これは、7.3 節と同じ理由で、入力の区間幅が小さくなると、方法 1 と方法 2 の極値の計算での区間拡大が大きくなるためであると考えられる。

8.5 入力が二変数の場合の性能評価

入力が多変数の場合の affine 演算と拡張 affine 演算の性能の違いを見るために、次の例を用いる。

$$\begin{aligned} f(x,y) &= xy\left(\frac{y}{x}-\frac{x}{y}\right)-y^2+x^2, \\ &\text{evaluate } f(x,y), \\ x &\in 10000+[-\varepsilon,\varepsilon], \\ y &\in 10001+[-\varepsilon,\varepsilon]. \end{aligned} \tag{26}$$

入力の区間半径と出力の区間半径との関係を図 7 に示す。affine 演算の乗算は [4] の最良乗算を用いている。

また、(26) で $\varepsilon = 10^{-1}$ として、二つの入力区間 $10000+[-\varepsilon,\varepsilon]$, $10001+[-\varepsilon,\varepsilon]$ を n 個に等分割したもの

をそれぞれ $x_1, x_2, \ldots, x_n$ と $y_1, y_2, \ldots, y_n$ とおく。そして、

$$\bigcup_{i,j=1}^{n} f(x_i, y_j)$$

を包含する区間を、affine 演算と拡張 affine 演算で計算する。計算される区間の半径が 5×10^{-7} 以下になるように分割数 n を決めるとすると、それぞれの演算について分割数と計算時間は表 9 のようになる。

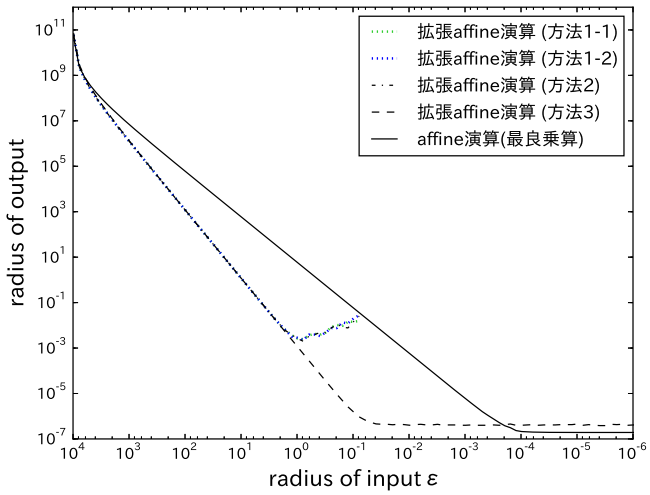


図 7: 入力が二変数の場合の性能評価の結果

表 9: 入力が二変数の場合の性能評価の結果 2

	分割数	計算時間 [μs]
affine 演算 ((1) による乗算)	688	3883867
affine 演算 (最良乗算)	495	12420933
拡張 affine 演算 (方法 3)	2	4453

結果から、affine 演算の方は入力の分割数が多くなり、計算時間の面で拡張 affine 演算の方が有利となることがわかる。affine 演算の分割数が多くなるのは入力が多変数になって区間拡大が起こりやすくなったためで、時間が大きく増加するのは入力が 2 つある場合は分割数の 2 乗に比例して計算量が増えるためだと考えられる。

方法 1 と方法 2 の出力の区間幅が途中で大きくなっているのは、8.4 節と同じ理由だと考えられる。

8.6 入力が三変数の場合の性能評価

入力が三変数になった場合の affine 演算と拡張 affine 演算の性能の違いを見るために、次の例を用いる。

$$\begin{aligned} f(x,y,z) &= xyz\left(\frac{yz}{x}-\frac{x}{y}\right)-y^2z^2+x^2z, \\ &\text{evaluate } f(x,y,z), \\ x &\in 10000+[-\varepsilon,\varepsilon], \\ y &\in 10001+[-\varepsilon,\varepsilon], \\ z &\in 10002+[-\varepsilon,\varepsilon]. \end{aligned} \tag{27}$$

入力の区間半径と出力の区間半径との関係を図 8 に示す。affine 演算の乗算は [4] の最良乗算を用いている。

また、(26) で $\varepsilon = 10^{-2}$ として、三つの入力区間 $10000+[-\varepsilon,\varepsilon]$, $10001+[-\varepsilon,\varepsilon]$, $10002+[-\varepsilon,\varepsilon]$ を n 個に等分割したものをそれぞれ $x_1, x_2, \ldots, x_n$ と $y_1, y_2, \ldots, y_n$ と $z_1, z_2, \ldots, z_n$ とおく。そして、

$$\bigcup_{i,j,k=1}^{n} f(x_i, y_j, z_k)$$

を包含する区間を、affine 演算と拡張 affine 演算で計算する。計算される区間の半径が 3×10^1 以下になるように分割数 n を決めるとすると、それぞれの演算について分割数と計算時間は表 10 のようになる。

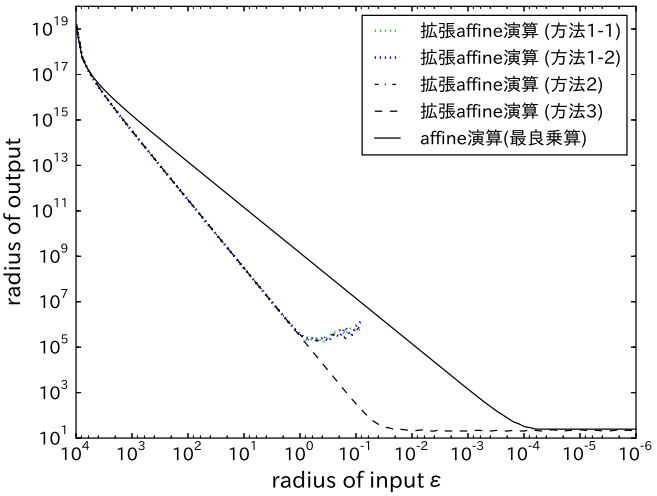


図 8: 入力が三変数の場合の性能評価の結果

表 10: 入力が三変数の場合の性能評価の結果 2

	分割数	計算時間 [μs]
affine 演算 ((1) による乗算)	246	236016083
拡張 affine 演算 (方法 3)	1	5689

8.4 節と 8.5 節の結果から、affine 演算の最良乗算は (1) による乗算の数倍遅くなることが予想され、探索に非常に時間がかかると考えたので計算は行わなかった。

結果から、affine 演算の方は入力の分割数が多くなり、計算時間の面で拡張 affine 演算の方が有利となることがわかる。affine 演算の分割数が多くなるのは入力が多変数になって区間拡大が起こりやすくなったためで、時間が大きく増加するのは入力が 3 つある場合は分割数の 3 乗に比例して計算量が増えるためだと考えられる。また、入力が一変数や二変数の場合と比較すると、入力変数が増えるほど拡張 affine 演算の方が有利になることがわかる。

方法 1 と方法 2 の出力の区間幅が途中で大きくなっているのは、8.4 節と同じ理由だと考えられる。

9　まとめと今後の課題

拡張 affine 演算における四則演算の計算方法と必要なアルゴリズムを考案し、ライブラリとして実装することができた。また、数値実験で、出力に一定の精度が求められていて入力を分割しなければいけない状況では拡張 affine 演算が有用な場合があり、特に入力が多変数であれば拡張 affine 演算が非常に高速となる場合があることが示せた。

拡張 affine 演算の計算時間は、二次項を二次形式を使って保持しているので、ダミー変数の個数の 2 乗に比例して増加する。ダミー変数は非線形な演算をする度に増加するため、計算時間の増加をできる限り抑えられるように、計算方法の工夫が必要だと考えられる。数値実験より、状況によって affine 演算と拡張 affine 演算のどちらが有利になるかが変わることがわかったので、両者のいいところを取ることで改善を図るということも考えられる。また、kv ライブラリの affine 演算には区間拡大をできるだけ抑えつつダミー変数の数を減らすような処理が実装されており、同様なダミー変数の数を減らす方法を考案・実装することで計算時間の増大を抑えるということも考えられる。

本論文では四則演算しか実装できていないので、実際の計算に利用するには、平方根などの計算も実装する必要がある。

10　謝辞

本論文の執筆にあたって、山本野人教授と山崎匡准教授にご指導をいただいたことを感謝いたします。また、数値実験の比較の対象とした kv ライブラリを開発した早稲田大学の柏木雅英教授に感謝いたします。

参考文献

[1] 中尾充宏・山本野人,「精度保証付き数値計算」, 日本評論社, 1998.

[2] 大石進一,「精度保証付き数値計算 (現代非線形科学シリーズ)」, コロナ社, 2000.

[3] 柏木雅英, 「Affine Arithmetic について」, http://verifiedby.me/kv/affine/affine.pdf, 2017/12/25 アクセス.

[4] 宮島信也,「区間解析における関数の値域の評価に関する研究」, 2004 年度早稲田大学博士論文, 2005.

[5] Siegfried M. Rump and Masahide Kashiwagi, *Implementation and improvements of affine arithmetic*, Nonlinear Theory and Its Applications, IEICE, Vol. 6, No. 3, pp. 341–359, 2015.

[6] E. W. Cheney,「近似理論入門」一松信・新島耕一訳, 共立出版, 1977.

[7] L. Veidinger, *On the numerical determination of the best approximations in the Chebyshev sense*, Numerische Mathematik, Vol. 2, Issue 1, pp. 99–105, 1960.

[8] 吉田年雄,「数値解析の基礎・基本 (理工系数学の基礎・基本)」, 牧野書店, 2005.

[9] W. H. Press 他著,「ニューメリカルレシピ・イン・シー 日本語版 C 言語による数値計算のレシピ」丹慶勝市 他訳, 技術評論社, 1993.

[10] Boost, *Boost C++ Libraries*, http://www.boost.org/.

[11] 柏木雅英,「kv - a C++ Library for Verified Numerical Computation」, http://verifiedby.me/kv/, 2018/1/12 アクセス.

[12] S. M. Rump, *INTLAB - INTerval LABoratory - Reliable Computing - TUHH*, http://www.ti3.tu-harburg.de/rump/intlab/, 2018/1/12 アクセス.